SWMM 5.1 Co-simulation Toolbox - User's Manual

Universidad de los Andes - Department of Electrical and Electronic Engineering

Contents

1 Introduction

SWMM is an open source platform that is used for planning, analysis and design of related to stormwater runoff, combined and sanitary sewers, and other drainage systems in urban areas. Its mode of operation is to display an interface where a network can be modelled, using objects such as conduits, storage units, subcatchments, among others. The user sets attributes for each element in the network and then executes the calculation module of SWMM.

Due to the changes that the design of sewer systems is suffering today, because of the inclusion of actuators, control rules, and optimal design, SWMM has become a limited tool; despite of its rule-based control module. However, it has a big potential being an open source software. Thanks to this, it is possible to enhance its functionality in order to include new methods that allow its users to simulate optimization and control processes.

Consequently, a new toolbox for SWMM 5.1 has been developed, bringing new functionality to the program, as it was described before. This toolbox works as a co-simulation engine that retrieves information from SWMM during the simulation and allows to modify initial conditions, and attributes like an orifice/valve setting or the maximum volume of an storage unit.

Furthermore, in order to make this tool scalable and multidisciplinary, it has been developed for three programming languages: Python, Matlab and LabVIEW. The syntax for each tool is very similar, so it is easy to migrate from one to another depending on the type of project that is wanted to be developed. It is worth to say that this toolbox requires a little effort to be learnt if the user has worked with SWMM before. Nevertheless, some knowledge of the programming language is required.

That said, this document will cover six main topics that are described below.

1. *How does SWMM work?* - The main goal of this section is to clarify doubts related to SWMM file processing and explain, with a simplistic perspective, the calculation algorithm of SWMM; which is necessary to understand how the toolbox functions work, and how to use them for a co-simulation project.

2. *How does the Toolbox work?* - General description of the Toolbox.

3. *Co-simulation functions and constants* - Deep explanation of each of the functions and constants included in the toolbox.

4. *Example running a single simulation.* - the example is developed for the three programming languages in order to compare the ways to declare a function in each one.

5. *Example running n simulations.*

2 How does SWMM work?

For the purpose of this manual, the complete explanation of the computational methods and the base model used by SWMM is not going to be explained; it can be checked in the SWMM 5 User's Manual which is available on the SWMM Website. However, it is important to note that SWMM is a phisically based, discrete-time simulation model that employs principles of conservation of mass, energy and momentum [**?**]. As a discrete-time tool its algorithm finds n solutions, for the flow routing problem, in n time intervals for a time window and initial conditions defined by the user. With that in mind, it is possible to understant now how SWMM makes its calculations checking the whole process; there are seven subprocesses which make up the simulation algorithm as it is shown in Figure 1.

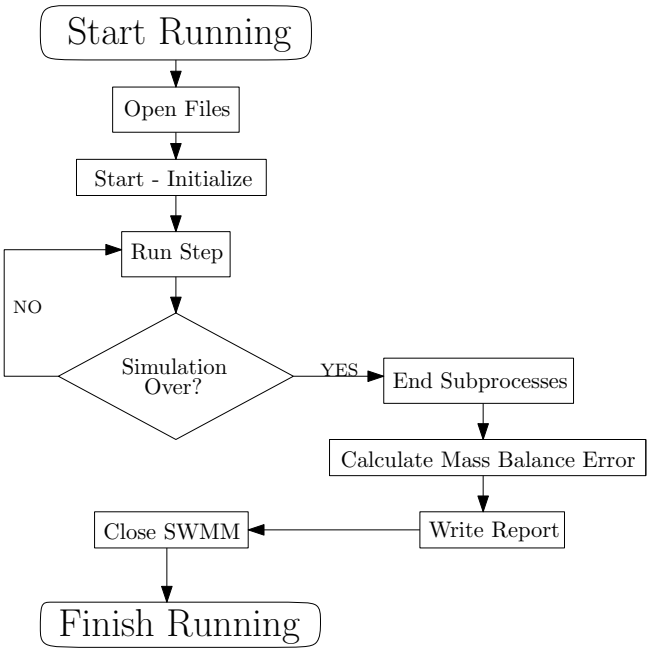

Figure 1. Algorithm to run a simulation in SWMM.

1. Open SWMM files: SWMM opens three files along the simulation, the input file (.inp) that contains all the information about the model, the report file (.rpt) which contains a status report of the results of a run and the output file (.out) that is a binary file with the numerical results of a succesfull run.

2. Start Simulation: the internal parameters of the program are initialized.
3. Run Step: advances the simulation by one routing time step.
4. End Subprocesses: all the computing systems and subprocesses of SWMM are ended.
5. Calculate Mass Balance Error: at the end of a simulation SWMM calculates three errors based on the mass conservation principle, the runoff error, the flow routing error and the quality routing error.
6. Write Report: SWMM writes the information for the output and report files.
7. Close SWMM: the files used by SWMM are closed. The simulation is over.

3 How does the Toolbox work?

This toolbox has been created as an additional module of the SWMM computational engine in order to preserve the code integrity. It has been compiled as a DLL, so it can be compatible with C-based programming languages. Additionally, some of the DLL functions that were created by the US-EPA were reused. Taking this into consideration, in order to enhance the toolbox functionality, following the style that has been used so far, you can download the original code of SWMM from the SWMM Website and the toolbox C files from the Toolbox Repository.

So far only three new functions have been developed. These can be used for two different kinds of projects: running a SWMM simulation modifying the percentage of openning of an orifice and running several SWMM simulations in series, retrieving information and modifying initial conditions. At the end of this document, there are two examples that explain better what have been stated before.

4 Co-simulation functions and constants

The toolbox is composed of several functions that can reproduce a SWMM simulation with control actions and optimal model attributes. It also has useful constants that simplify the code development process. In the tables below both, general functions and constants, are described deeply.

Nevertheless, some of the return variables of the functions vary depending on the programming language. For instance, the "Run step" function for the Python Toolbox returns nothing, it just advances the simulation by one step, while the same function for the Matlab Toolbox returns the value of the current hour of simulation. This variations are explained better in the next section, where the differences between Toolboxes, secondary functions, and instantiation of methods are explained better.

Table 1. Functions and their parameters with descriptions.

Functions		**Parameters**	
Open File	Opens the files required to run a SWMM simulation	*Input file*	Path to the input file.
		Report file	Path to the report file.
		Output file	Path to the output file.
Start	Starts a SWMM simulation.	*Write Report*	Constant related to the report writing.
Run Step	Advances the simulation by one routing time step.		
End	Ends a SWMM simulation.		
Close	Closes a SWMM project.		
Get	Returns the value of an object attribute.	*Object type*	Constant related to the type of object.
		Object ID	ID of the object.
		Attribute	Constant related to the type of attribute.
		Unit System	Constant related to the units of the attribute.
Modify Setting	Modifies the setting of an orifice during the simulation.	*Orifice ID*	ID of the orifice.
		New setting	New value for the orifice setting (decimal value).
Modify Input	Modifies a specific attribute from the input file.	*Filename*	Path to the input file.
		Object ID	ID of the object.
		Object type	Constant related to the type of the object in the input file.
		Attribute	Constant related to the attribute of the object in the input file.
		Value	New value for the attribute in the input file.
Mass Bal. Error	Gets the mass balance errors of the simulation.		

Table 2. Constants with their descriptions.

Types of objects	
SUBCATCH	Subcatchment object
NODE	Node Object
LINK	Link object
Unit System	
US	English unit system
SI	International unit system
DIMENSIONLESS	Used if an attribute is dimentionless (e.g. Froude number)
Type of Attributes	
DEPTH	Max. depth of links and nodes.
VOLUME	Volumetric capacity of links and nodes.
FLOW	Link flow.
SETTING	Orifice setting - percentage of openning (decimal number).
FROUDE	The Froude number in a link object.
INFLOW	The inflow of a node object.
FLOODING	Flood volume of a node.
PRECIPITATION	Intensity of precipitation in a subcatchment.
RUNOFF	Runoff in a subcatchment.
Report Options	
NO_REPORT	The simulation results are not written in the report file.
WRITE_REPORT	The simulation results are written in the report file.
Input file	
INVERT	Invert elevation of a node.
DEPTH_SIZE	Depth of links, junctions and storage units.
STORAGE_A	A - Geometric parameter of an storage unit.
STORAGE_B	B - Geometric parameter of an storage unit.
STORAGE_C	C - Geometric parameter of an storage unit.
LENGTH	The length of a link.
ROUGHNESS	Roughness of a link.
IN_OFFSET	Inlet offset of a conduit.
OUT_OFFSET	Outlet offset of a conduit.
JUNCTION	Junction object (It belongs to the node type)
STORAGE	Storage object (It belongs to the node type)

5 Example running a single simulation

The case study

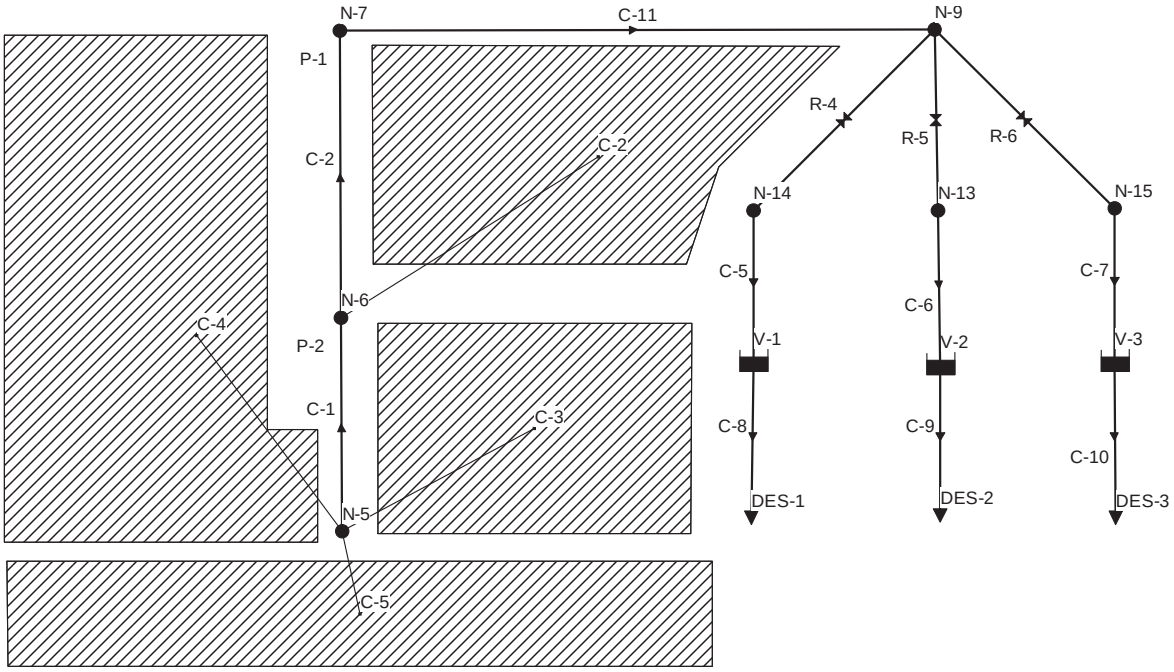

Figure 2. Case study used as an example to show the functionality of the toolbox

The model used for this example is available in the Toolbox Repository. Basically, it is composed by four subcatchments that are affected by a precipitation event, the precipitation becomes runoff, and it is transported through a system of channels that is divided in three. At the beginning of the bifurcation there are three gates, modelled as orifices, controlling the passage of rainwater in order to distribute it efficiently in three tanks of different volumes.

In this example it is going to be shown how to use the functions of the Toolbox to reproduce the Running Algorithm of SWMM, adding the functionality of retrieving and modifying parameters during the simulation.

The Python Toolbox - (Check the source code here)

```
# Imported modules
from swmm import *  # The SWMM module
import matplotlib.pyplot as plt  # Module for plotting

# ****************************************************************************
#  Declaration of simulation files and variables
# ****************************************************************************

inp    = "swmm_files/3tanks.inp"  # Input file
report = "swmm_files/3tanks.rpt"  # Report file
out    = "swmm_files/3tanks.out"  # Output file
flow   = []
vol    = []
time   = []

# ****************************************************************************
#  Initializing SWMM
# ****************************************************************************

open_file(inp, report, out)  # Step 1
start(NO_REPORT)  # Step 2

# ****************************************************************************
#  Step Running
# ****************************************************************************

# Main loop: finished when the simulation time is over.
while( not is_over() ):

  # ----------------- Run step and retrieve simulation time -----------

  time.append(get_time())
  run_step()  # Step 3

  # --------- Retrieve & modify information during simulation ---------
  # Retrieve information about flow in C-5
  f = get(LINK, 'C-5', FLOW, SI)
  # Stores the information in the flow vector
  flow.append(f)
  # Retrieve information about volume in V-1
  v = get(NODE, 'V-1', VOLUME, SI)
  # Stores the information in the volume vector
  vol.append(v)
```

```
    # -------------------------- Control Actions ----------------------
46
    # If the flow in C-5 is greater or equal than 2 m3/s the setting
    # upstream of the link is completely closed, else it is completely
    # opened.

51  if f >= 2:
      modify_setting('R-4', 0)
    else:
      modify_setting('R-4', 1)

56  # ***************************************************************************
    #  End of simulation
    # ***************************************************************************

    end()  # Step 4
61  errors = get_mass_bal_error()  # Step 5
    save_report()  # Step 6
    close()  # Step 7

    # ***************************************************************************
66  #  Interacting with the retrieved data
    # ***************************************************************************

    print "\n Runoff error: %.2f %%\n \
      Flow routing error: %.2f %%\n \
71    Quality routing error: %.2f %%\n" % (errors[0], errors[1], errors[2])

    plt.plot(time, flow)
    plt.title('C-5 flow [m3/s]')
    plt.xlabel('Time [hours]')
76  plt.show()
```

After checking the code you can notice that it has been divided in sections, so it is possible to relate each section to a part of the Running Algorithm. Additionally, every step related to the Running Algorithm has been marked with its corresponding number; it is important to respect this order. It is also important to be aware of the functions and constants declaration. In this case, all the constants and functions of the SWMM module have been imported. However, if the module had been imported as a single component, all the functions should have been declared instantiating the module.

```
swmm.start(swmm.NO_REPORT) # e.g.
```

Now, checking the code in more detail there are additional functions related to the Python Toolbox:

- The *is_over*() function: it has been created in order to determine when the simulation is over. It returns *True* if there are no more steps left to run.
- The *get_time*() function: it returns the current simulation time in hours. It is important to notice that is has been declared before the *run_step*() function. This is not a coincidence, it is important to do this if you want to get a coherent time vector at the end of the simulation. If you declare it after *run_step*(), in the last step the time value is going to be zero, because zero is the value generated by *run_step*() to warn that the simulation is over.

Besides this, the syntax to retrieve and modify information is pretty simple and the control actions can be as complicated as Python coding is. The last important thing is that the *get_mass_bal_error*() function returns a tuple with the values of each error. The first value in the tuple is the runoff error, the second is the flow routing error and the last is the quality routing error.

The results obtained running the example code are shown in Figure 3. As you can see, the rule-based control action modifies the valve setting maintaining the flow in C-5 is relatively close to 2 m^3/s.

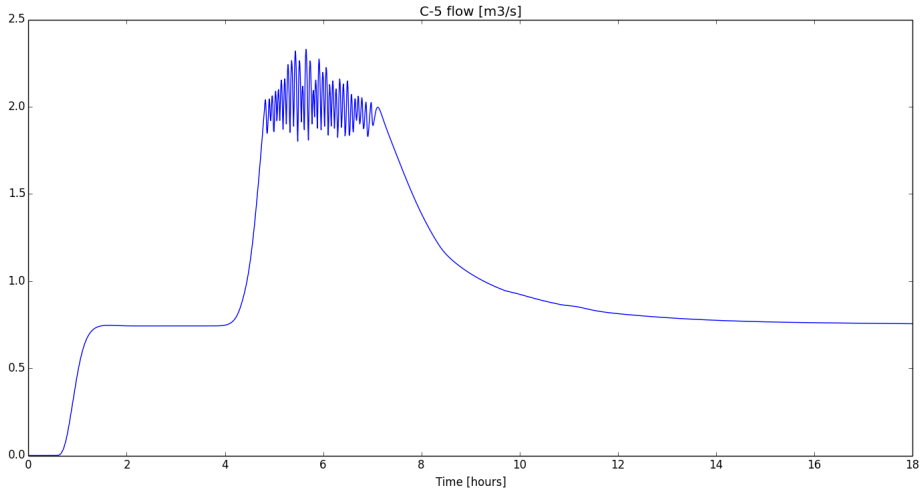

Figure 3. Python example results.

The Matlab Toolbox - (Check the source code here)

```
% Get the toolbox constants and variables
swmm_get_constants;

% ************************************************************************
%  Declaration of simulation files and variables
% ************************************************************************

inp     = 'swmm_files/3tanks.inp';  % Input files
report  = 'swmm_files/3tanks.rpt';  % Report file
out     = 'swmm_files/3tanks.out';  % Output file
i       = 1;  % Index for vectors

% ************************************************************************
%  Initializing SWMM
% ************************************************************************

swmm_open(inp, report, out); % Step 1
swmm_start(NO_REPORT); % Step 2

% ************************************************************************
%  Step Running
% ************************************************************************

% Main loop: finished when the simulation time is over.
% The simulation is over when elapsed_time == 0.
while elapsed_time ~= 0

    % ----------------- Run step and retrieve simulation time -----------

    time(i,:) = elapsed_time;
    elapsed_time = swmm_step; % Step 3

    % --------- Retrieve & modify information during simulation ---------
    % Retrieve information about flow in C-5 and volume in V-1
    flow(i,:) = swmm_get(LINK, 'C-5', FLOW, SI);
    volume(i,:) = (NODE, 'V-1', VOLUME, SI);

    % -------------------------- Control Actions -----------------------

    % If the flow in C-5 is greater or equal than 0.015 m3/s the setting
    % upstream of the link is completely closed, else it is completely
    % opened.
```

```
        if flow(i,:) >= 0.015
            swmm_modify_setting('R-4', 0);
        else
            swmm_modify_setting('R-4', 1);
        end

        i = i+1;

    end

    % ************************************************************************
    %  End of simulation
    % ************************************************************************

    swmm_end; % Step 4
    errors = swmm_massBalErr; % Step 5
    swmm_report; % Step 6
    swmm_close; % Step 7

    % ************************************************************************
    %  Interacting with the retrieved data
    % ************************************************************************

    disp(sprintf('Runoff error: %.2f%%'    , errors(1)));
    disp(sprintf('Hydrologic error: %.2f%%', errors(2)));
    disp(sprintf('Quality error: %.2f%%\n' , errors(3)));

    plot(time, flow);
```

The Matlab example code has been divided in sections so each section is related with a part of the SWMM Running Algorithm. Additionally, each step of the Running Algorithm has been marked and enumerated; it is necessary to respect this order. However there are special functions and variables for the Matlab module that have not been specified yet, these are:

- The *swmm_get_constants* function: it is used to declare all the constants and variables used by some of the Toolbox functions. After this instantiation, the constants can be declared as shown in the code, i.e. the constants are characterized by being written in uppercase.
- The *elapsed_time* variable: it is used to save the value of the simulation time, which is necessary to know when the simulation is over, because when it is equal to zero there are no steps left to be run. Moreover, this variable is initialized after calling the

swmm_get_constants function, so you do not have to be worried about it. Keep in mind that this variable is refreshed using the *swmm_step* function as shown in line 31 of the example code.

Finally, it is worth to say that the *swmm_massBalErr* function returns the errors, related to the mass balance testing, as a vector with three elements. The first one is the runoff error, the second is the flow routing error and the third is the quality error.

The results obtained after the simulation are shown in Figure 4. Because of the control actions, the setting of R-4 remains close to 2 m^3/s.

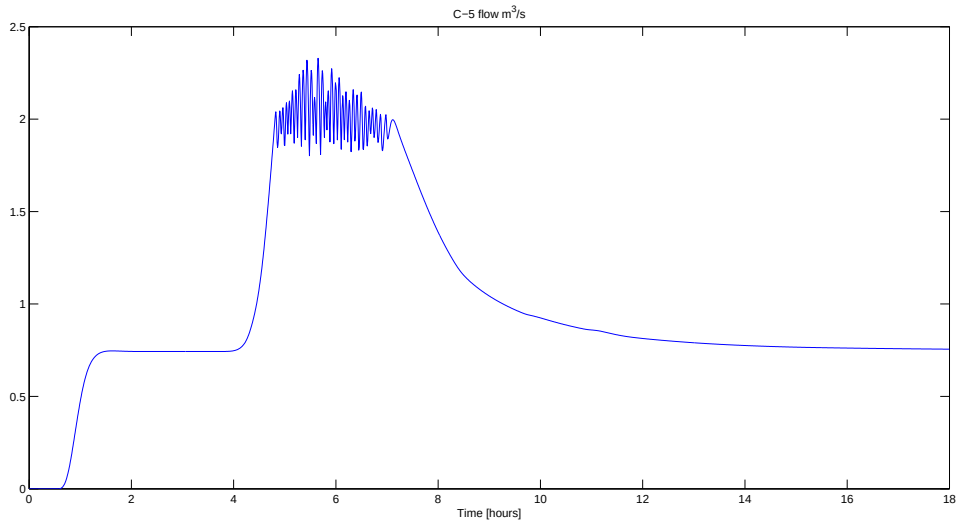

Figure 4. Matlab example results.

The LabVIEW Toolbox

6 Example running n simulations